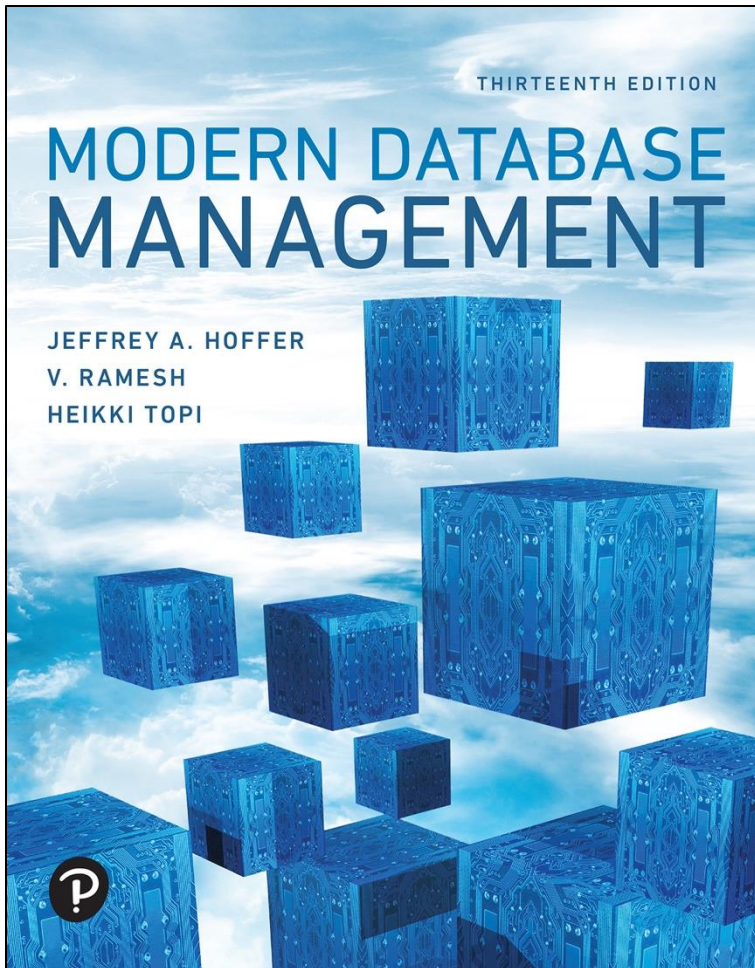


Modern Database Management

Thirteenth Edition



Chapter 6

Advanced SQL

Learning Objectives

6.1 Define terms

6.2 Write single- and multiple-table queries using SQL commands

6.3 Define three types of join commands and use SQL to write these commands

6.4 Write noncorrelated and correlated subqueries and know when to write each

6.5 Write queries to create dynamic and materialized views

6.6 Understand common uses of database triggers and stored procedures

6.7 Discuss the SQL:2011 and SQL:2016 standards and explain SQL enhancements and extensions

Processing Multiple Tables (1 of 2)

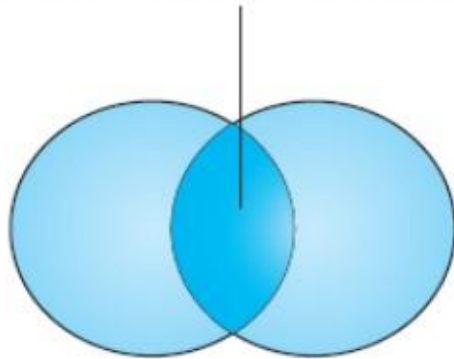
- Join
 - A relational operation that causes two or more tables with a common domain to be combined into a single table or view
- Equi-join
 - A join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- Natural (inner) join
 - An equi-join in which one of the duplicate columns is eliminated in the result table

Processing Multiple Tables (2 of 2)

- Outer join
 - A join in which rows that do **not** have matching values in common columns are nonetheless included in the result table (as opposed to **inner** join, in which rows must have matching values in order to appear in the result table)
- Union join
 - Includes all data from each table that was joined

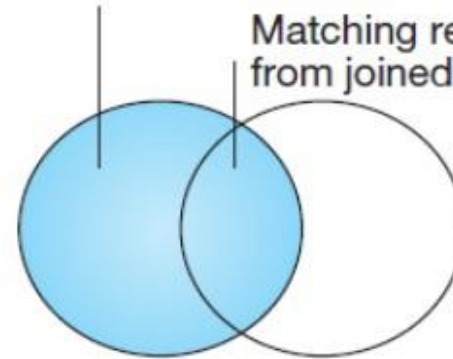
Figure 6-2 Visualization of Different Join Types, With the Results Returned in the Shaded Area

Darker area is result returned.



Natural Join

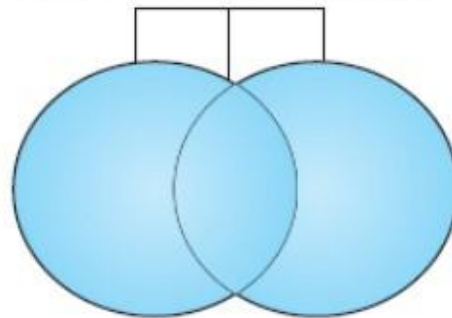
All records returned from outer table.



Matching records returned from joined table.

Left Outer Join

All records are returned.



Union Join

The Following Slides Create Tables for This Enterprise Data Model

(from Chapter 1, Figure 1-3)

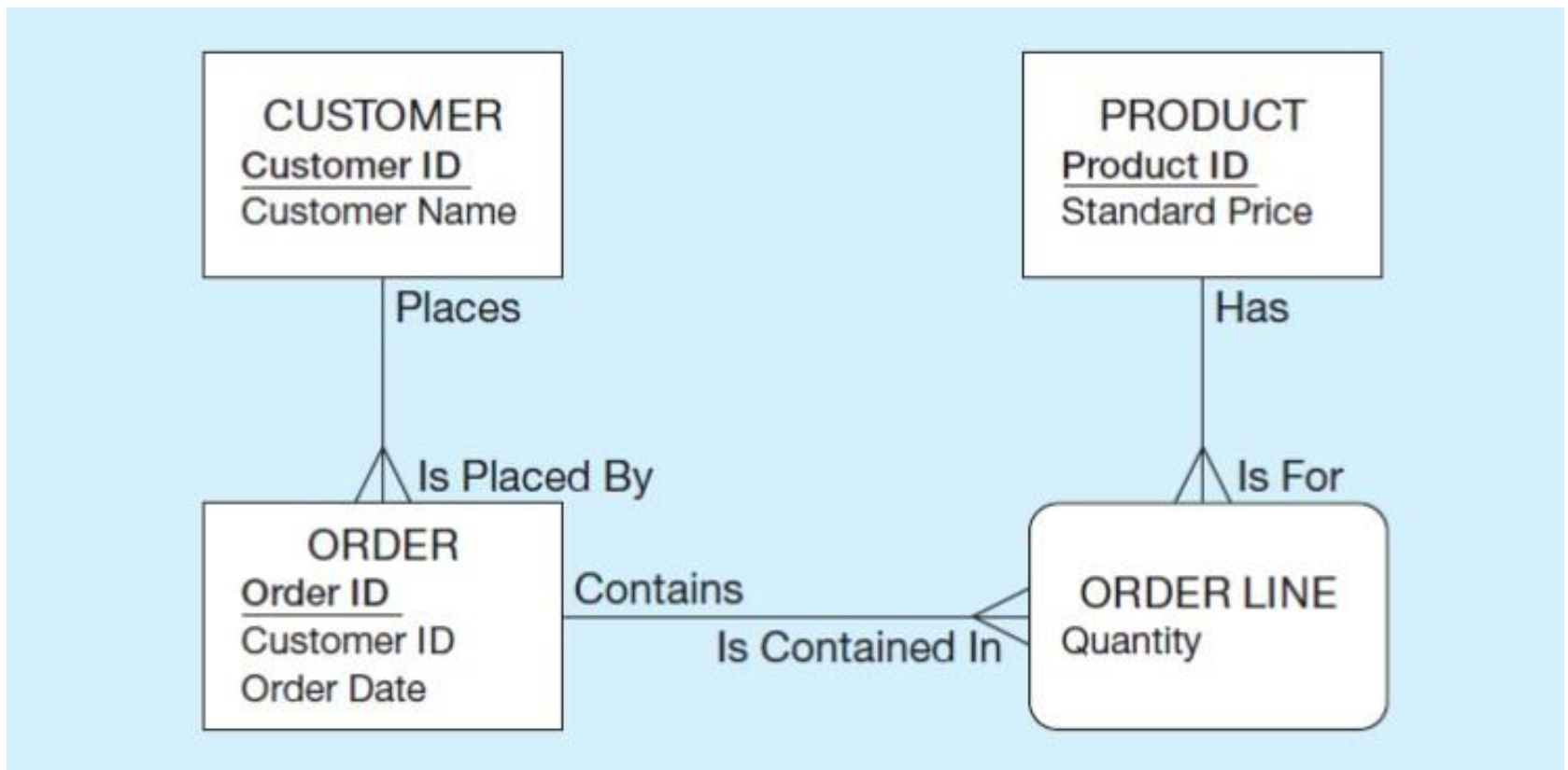


Figure 6-1 Pine Valley Furniture Company Customer_T and Order_T Tables, With Pointers From Customers to Their Orders

With pointers from orders to the customers who placed them

The image shows two database tables side-by-side. The left table is 'Order_T' and the right table is 'Customer_T'. Arrows point from the 'CustomerID' column in the 'Order_T' table to the 'CustomerID' column in the 'Customer_T' table, showing the relationship between orders and the customers who placed them.

OrderID	OrderDate	CustomerID
1001	10/21/2021	1
1002	10/21/2021	8
1003	10/22/2021	15
1004	10/22/2021	5
1005	10/24/2021	3
1006	10/24/2021	2
1007	10/27/2021	11
1008	10/30/2021	12
1009	11/5/2021	4
1010	11/5/2021	1
*	0	0

CustomerID	CustomerName	CustomerAddress	CustomerCit	CustomerStat	CustomerPostalCod
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-2871
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-7743
3	Home Furnishings	1900 Allard Ave.	Albany	NY	12209-1125
4	Eastern Furniture	1925 Beltline Rd.	Carteret	NJ	07008-3188
5	Impressions	5585 Westcott Ct.	Sacramento	CA	94206-4056
6	Furniture Gallery	325 Flatiron Dr.	Boulder	CO	80514-4432
7	Period Furniture	394 Rainbow Dr.	Seattle	WA	97954-5589
8	California Classics	816 Peach Rd.	Santa Clara	CA	96915-7754
9	M and H Casual Furniture	3709 First Street	Clearwater	FL	34620-2314
10	Seminole Interiors	2400 Rocky Point Dr.	Seminole	FL	34646-4423
11	American Euro Lifestyles	2424 Missouri Ave N	Prospect Park	NJ	07508-5621
12	Battle Creek Furniture	345 Capitol Ave. SW	Battle Creek	MI	49015-3401
13	Heritage Furnishings	66789 College Ave.	Carlisle	PA	17013-8834
14	Kaneohe Homes	112 Kiowai St.	Kaneohe	HI	96744-2537
15	Mountain Scenes	4132 Main Street	Ogden	UT	84403-4432
*	(New)				

Equi-Join Example

```
SELECT Customer_T.CustomerID, Order_T.CustomerID,  
       CustomerName, OrderID  
FROM Customer_T, Order_T  
WHERE Customer_T.CustomerID = Order_T. CustomerID  
ORDER BY OrderID
```

Result:

Customerid	Customerid	Customername	Orderid
	1	1 Contemporary Casuals	1001
	8	8 California Classics	1002
	15	15 Mountain Scenes	1003
	5	5 Impressions	1004
	3	3 Home Furnishings	1005
	2	2 Value Furniture	1006
	11	11 American Euro Lifestyles	1007
	12	12 Battle Creek Furniture	1008
	4	4 Eastern Furniture	1009
	1	1 Contemporary Casuals	1010

10 rows selected.

What are the customer IDs and names of all customers, along with the order IDs for all the orders they have placed?

Equi-Join Example – Alternative Syntax

An INNER Join

```
SELECT Customer_T.CustomerID, Order_T.CustomerID,  
       CustomerName, OrderID  
FROM Customer_T INNER JOIN Order_T ON  
       Customer_T.CustomerID = Order_T.CustomerID  
ORDER BY OrderID;
```

INNER JOIN clause is an alternative to WHERE clause, and is used to match primary and foreign keys.

An INNER join will only return rows from each table that have matching rows in the other.

This query produces the same results as the previous equi-join example.

Outer-Join Example

List the customer name, ID number, and order number for all customers. Include customer information even for customers that do not have an order.

```
SELECT Customer_T.CustomerID, CustomerName, OrderID  
FROM Customer_T LEFT OUTER JOIN Order_T  
WHERE Customer_T.CustomerID = Order_T.CustomerID;
```

LEFT OUTER JOIN clause causes rows from the first mentioned table (customer) to appear even if there is no corresponding order data.

Unlike an INNER join, this will include customer rows with no matching order rows.

For the tables in figure 6.1, this will return 16 rows. That's because there are 15 customers, and one of these customers has 2 orders.

Result

Note two rows for customer #1 Contemporary Casuals.

Also note that several customers don't have orders.

This is because of the left outer join.

Customerid	Customername	Orderid
1	Contemporary Casuals	1001
1	Contemporary Casuals	1010
2	Value Furniture	1006
3	Home Furnishings	1005
4	Eastern Furniture	1009
5	Impressions	1004
6	Furniture Gallery	-
7	Period Furniture	-
8	California Classics	1002
9	M & H Casual Furniture	-
10	Seminole Interiors	-
11	American Euro Lifestyles	1007
12	Battle Creek Furniture	1008
13	Heritage Furnishings	-
14	Kaneohe Homes	-
15	Mountain Scenes	1003
		-

16 rows selected.

Multiple Table Join Example

Assemble all information necessary to create an invoice for order number 1006.

Each pair of tables requires an equality-check condition in the WHERE clause, matching primary keys against foreign keys.

```
SELECT Customer_T.CustomerID, CustomerName, CustomerAddress,  
       CustomerCity, CustomerState, CustomerPostalCode, Order_T.OrderID,  
       OrderDate, OrderedQuantity, ProductDescription, StandardPrice,  
       (OrderedQuantity * ProductStandardPrice)  
FROM Customer_T, Order_T, OrderLine_T, Product_T  
WHERE Order_T.CustomerID = Customer_T.CustomerID  
       AND Order_T.OrderID = OrderLine_T.OrderID  
       AND OrderLine_T.ProductID = Product_T.ProductID  
       AND Order_T.OrderID = 1006;
```

Figure 6-4 Results From a Four-Table Join (Edited for Readability)

CUSTOMERID	CUSTOMERNAME	CUSTOMERADDRESS	CUSTOMER CITY	CUSTOMER STATE	CUSTOMER POSTALCODE
2	Value Furniture	15145 S. W. 17th St.	Plano	TX	75094 7743
2	Value Furniture	15145 S. W. 17th St.	Plano	TX	75094 7743
2	Value Furniture	15145 S. W. 17th St.	Plano	TX	75094 7743

ORDERID	ORDERDATE	ORDERED QUANTITY	PRODUCTNAME	PRODUCT STANDARDPRICE	(QUANTITY* STANDARDPRICE)
1006	24-OCT-21	1	Entertainment Center	650	650
1006	24-OCT-21	2	Writer's Desk	325	650
1006	24-OCT-21	2	Dining Table	800	1600

All rows returned from this query will pertain to OrderID 1006.

Note that the full query results include columns from four different tables.

Self Join Example

What are the employee ID and name of each employee and the name of his or her supervisor (label the supervisor's name Manager)?

```
SELECT E.EmployeeID, E.EmployeeName, M.EmployeeName AS Manager
FROM Employee_T E, Employee_T M
WHERE E.EmployeeSupervisor = M.EmployeeID;
```

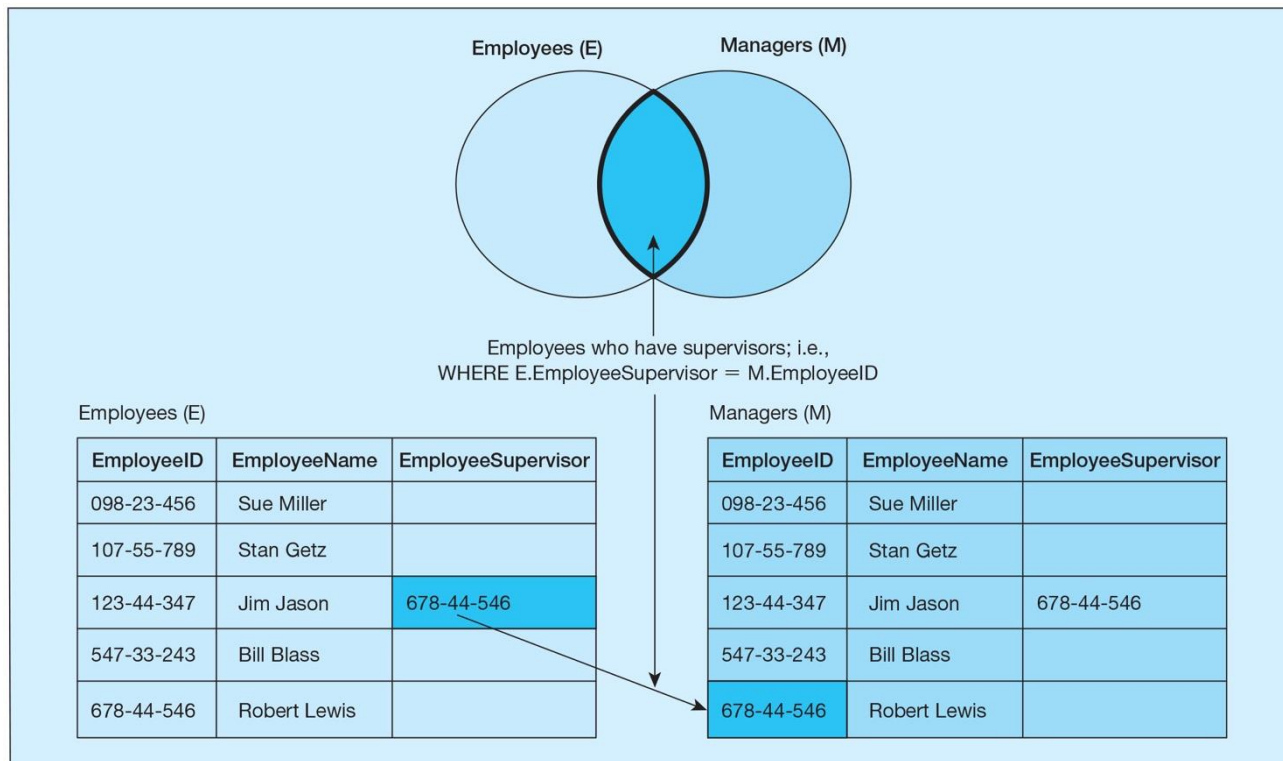
Result:

Employeeid	Employeename	Manager
123-44-347	Jim Jason	Robert Lewis

The same table is used on both sides of the join; distinguished using table aliases. See the next slide for details.

Figure 6-5 Example of a Self-Join

Self join involve tables that implement 1-to-many unary relationships.



Subqueries

- Subquery – placing an inner query (SELECT statement) inside an outer query
- Options:
 - In a condition of the WHERE clause
 - As a “table” of the FROM clause
 - Returning a field for the SELECT clause
 - Within the HAVING clause
- Subqueries can be:
 - Noncorrelated – executed once for the entire outer query
 - Correlated – executed once for each row returned by the outer query

Subquery Example

What are the name and address of the customer who placed order number 1008?

```
SELECT CustomerName, CustomerAddress, CustomerCity, CustomerState,  
CustomerPostalCode  
FROM Customer_T  
WHERE Customer_T.CustomerID =  
    (SELECT Order_T.CustomerID  
     FROM Order_T  
     WHERE OrderID = 1008);
```

Alternative Approach, Using a Join

What are the name and address of the customer who placed order number 1008?

```
SELECT CustomerName, CustomerAddress, CustomerCity,  
       CustomerState, CustomerPostalCode  
FROM Customer_T, Order_T  
WHERE Customer_T.CustomerID = Order_T.CustomerID  
       AND OrderID = 1008;
```

Figure 6-6 Graphical Depiction of Two Ways to Answer a Query With Different Types of Joins (1 of 2)

a) Join query approach

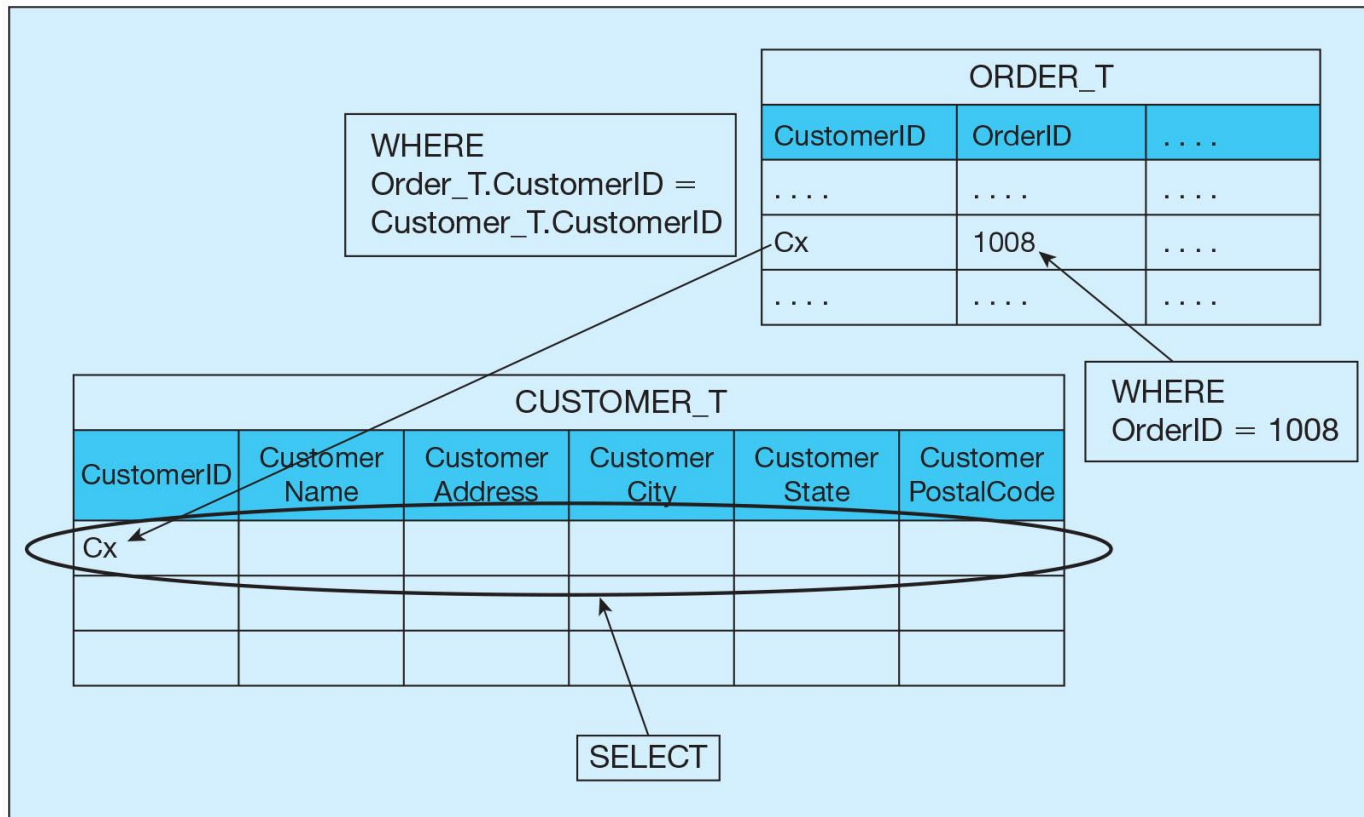
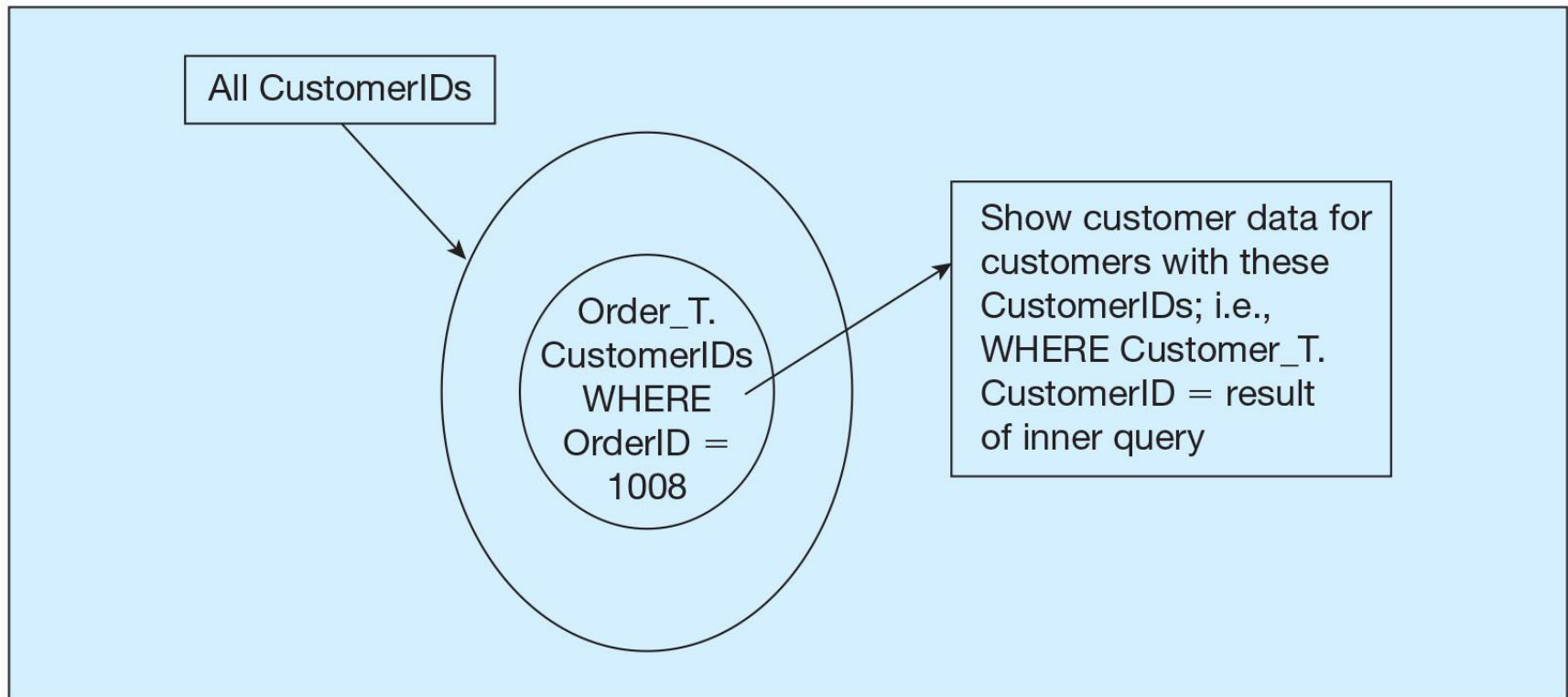


Figure 6-6 Graphical Depiction of Two Ways to Answer a Query With Different Types of Joins (2 of 2)

b) Subquery approach



Correlated vs Noncorrelated Subqueries

- Noncorrelated subqueries:
 - Do not depend on data from the outer query
 - Execute once for the entire outer query
- Correlated subqueries:
 - Make use of data from the outer query
 - Execute once for each row of the outer query
 - Can use the EXISTS and ALL operators

Example of a Correlated Subquery

List the details about the product with the highest standard price.

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
FROM Product_T PA
WHERE PA.ProductStandardPrice > ALL
      (SELECT ProductStandardPrice FROM Product_T PB
       WHERE PB.ProductID != PA.ProductID);
```

Result:

Productdescription	Productfinish	Productstandardprice
Dining Table	Natural Ash	800

Another Correlated Subquery

What are the order IDs for all orders that have included furniture finished in natural ash?

```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
  (SELECT *
   FROM Product_T
   WHERE ProductID = OrderLine_T.ProductID
   AND Productfinish = 'Natural Ash');
```

A correlated subquery always refers to an attribute from a table referenced in the outer query.

Figure 6-8 Subquery Processing (1 of 2)

a) Processing a noncorrelated subquery

What are the names of customers who have placed orders?

```
SELECT CustomerName
      FROM Customer_T
      WHERE CustomerID IN
```

```
(SELECT DISTINCT CustomerID
  FROM Order_T);
```

1. The subquery (shown in the box) is processed first and an intermediate results table created:
2. The outer query returns the requested customer information for each customer included in the intermediate results table:

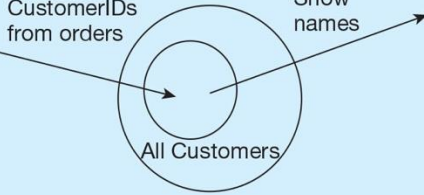
<table border="0"><thead><tr><th><u>CUSTOMERID</u></th></tr></thead><tbody><tr><td>1</td></tr><tr><td>8</td></tr><tr><td>15</td></tr><tr><td>5</td></tr><tr><td>3</td></tr><tr><td>2</td></tr><tr><td>11</td></tr><tr><td>12</td></tr><tr><td>4</td></tr></tbody></table> <p>9 rows selected.</p>	<u>CUSTOMERID</u>	1	8	15	5	3	2	11	12	4	<p>CustomerIDs from orders</p>  <p>Show names</p>	<table border="0"><thead><tr><th><u>CUSTOMERNAME</u></th></tr></thead><tbody><tr><td>Contemporary Casuals</td></tr><tr><td>Value Furniture</td></tr><tr><td>Home Furnishings</td></tr><tr><td>Eastern Furniture</td></tr><tr><td>Impressions</td></tr><tr><td>California Classics</td></tr><tr><td>American Euro Lifestyles</td></tr><tr><td>Battle Creek Furniture</td></tr><tr><td>Mountain Scenes</td></tr></tbody></table> <p>9 rows selected.</p>	<u>CUSTOMERNAME</u>	Contemporary Casuals	Value Furniture	Home Furnishings	Eastern Furniture	Impressions	California Classics	American Euro Lifestyles	Battle Creek Furniture	Mountain Scenes
<u>CUSTOMERID</u>																						
1																						
8																						
15																						
5																						
3																						
2																						
11																						
12																						
4																						
<u>CUSTOMERNAME</u>																						
Contemporary Casuals																						
Value Furniture																						
Home Furnishings																						
Eastern Furniture																						
Impressions																						
California Classics																						
American Euro Lifestyles																						
Battle Creek Furniture																						
Mountain Scenes																						

Figure 6-8 Subquery Processing (2 of 2)

b) Processing a correlated subquery

What are the order IDs for all orders that have included furniture finished in natural ash?

```

SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
  (SELECT *
   FROM Product_T
   WHERE ProductID = OrderLine_T.ProductID
   AND Productfinish = 'Natural Ash');
  
```

OrderID	ProductID	OrderedQuantity
1001	1	1
1001	2	2
1001	4	1
1002	3	3
1003	3	3
1004	5	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10
0	0	0

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	End Table	Cherry	\$175.00	10001
2	Coffee Table	Natural Ash	\$200.00	20001
3	Computer Desk	Natural Ash	\$375.00	20001
4	Entertainment Center	Natural Maple	\$650.00	30001
5	Writer's Desk	Cherry	\$325.00	10001
6	8-Drawer Dresser	White Ash	\$750.00	20001
7	Dining Table	Natural Ash	\$800.00	20001
8	Computer Desk	Walnut	\$250.00	30001
*	(AutoNumber)		\$0.00	

- The first order ID is selected from OrderLine_T: OrderID = 1001.
- The subquery is evaluated to see if any product in that order has a natural ash finish. Product 2 does, and is part of the order. EXISTS is valued as true and the order ID is added to the result table.
- The next order ID is selected from OrderLine_T: OrderID = 1002.
- The subquery is evaluated to see if the product ordered has a natural ash finish. It does. EXISTS is valued as true and the order ID is added to the result table.
- Processing continues through each order ID. Orders 1004, 1005, and 1010 are not included in the result table because they do not include any furniture with a natural ash finish. The final result table is shown in the text on page 264.

Derived Table (Subquery in the FROM Clause of the Outer Query)

What are the order I Ds for all orders that have included furniture finished in natural ash?

```
SELECT ProductDescription, ProductStandardPrice, AvgPrice
FROM
    (SELECT AVG(ProductStandardPrice) AvgPrice FROM Product_T),
    Product_T
WHERE ProductStandardPrice > AvgPrice;
```

Here, the subquery forms the derived table used in the FROM clause of the outer query. The AvgPrice column from the subquery is used in the SELECT clause of the outer query.

UNION — Combining Queries

Combine the output (union of multiple queries) together into a single result table

With UNION queries, the quantity and data types of the attributes in the SELECT clauses of both queries must be identical.

```
SELECT C1.CustomerID, CustomerName, OrderedQuantity,  
'Largest Quantity' AS Quantity  
FROM Customer_T C1,Order_T O1, OrderLine_T Q1  
WHERE C1.CustomerID = O1.CustomerID  
AND O1.OrderID = Q1.OrderID  
AND OrderedQuantity =  
(SELECT MAX(OrderedQuantity)  
FROM OrderLine_T)  
UNION  
SELECT C1.CustomerID, CustomerName, OrderedQuantity,  
'Smallest Quantity'  
FROM Customer_T C1, Order_T O1, OrderLine_T Q1  
WHERE C1.CustomerID = O1.CustomerID  
AND O1.OrderID = Q1.OrderID  
AND OrderedQuantity =  
(SELECT MIN(OrderedQuantity)  
FROM OrderLine_T)  
ORDER BY 3;
```

Result:

CustomerID	CustomerName	OrderedQuantity	Quantity
1	Contemporary Casuals	1	Smallest Quantity
2	Value Furniture	1	Smallest Quantity
1	Contemporary Casuals	10	Largest Quantity

Conditional Expressions Using Case Keyword

```
SELECT CASE
  WHEN ProductLine = 1 THEN ProductDescription
  ELSE '####'
END AS ProductDescription
FROM Product_T;
```

A CASE expression acts like an if-then statement. It allows you to choose what will appear in a column of the result set, depending on a condition.

Result:

Productdescription

End Table

####

####

####

Writers Desk

####

####

####

More Complicated SQL Queries

- Production databases contain hundreds or even thousands of tables, and tables could include hundreds of columns.
- So, sometimes query requirements can be very complex.
- Sometimes it's useful to combine queries, through the use of Views.
- If you use a view (which is a query), you could have another query that uses the view as if it were a table.

Using a View in Your Query

For each salesperson, list his or her biggest-selling product.

The view:

```
CREATE VIEW TSales AS
SELECT SalespersonName,
       ProductDescription,
       SUM(OrderedQuantity) AS Totorders
FROM Salesperson_T, OrderLine_T, Product_T, Order_T
WHERE Salesperson_T.SalespersonID=Order_T.SalespersonID
AND Order_T.OrderID=OrderLine_T.OrderID
AND OrderLine_T.ProductID=Product_T.ProductID
GROUP BY SalespersonName, ProductDescription;
```

The query using the view:

```
SELECT SalespersonName, ProductDescription
FROM TSales AS A
WHERE Totorders = (SELECT MAX(Totorders) FROM TSales B
WHERE B.SalespersonName = A.SalespersonName);
```

Tips for Developing Queries

- Be familiar with the data model (entities and relationships)
- Understand the desired results
- Know the attributes desired in results
- Identify the entities that contain desired attributes
- Review ERD
- Construct a WHERE equality for each link
- Fine tune with GROUP BY and HAVING clauses if needed
- Consider the effect on unusual data

Query Efficiency Considerations

- Instead of `SELECT *`, identify the specific attributes in the `SELECT` clause; this helps reduce network traffic of result set
- Limit the number of subqueries; try to make everything done in a single query if possible
- If data is to be used many times, make a separate query and store it as a view

Guidelines for Better Query Design (1 of 2)

- Understand how indexes are used in query processing
- Keep optimizer statistics up to date
- Use compatible data types for fields and literals
- Write simple queries
- Break complex queries into multiple simple parts
- Don't nest one query inside another query
- Don't combine a query with itself (if possible avoid self-joins)

Guidelines for Better Query Design (2 of 2)

- Create temporary tables for groups of queries
- Combine update operations
- Retrieve only the data you need
- Don't have the DBMS sort without an index
- Learn!
- Consider the total query processing time for ad hoc queries

Using and Defining Views

- Dynamic View
 - A “virtual table” created dynamically upon request by a user
 - No data actually stored; instead data from base table made available to user
 - Based on SQL SELECT statement on base tables or other views
- Materialized View
 - Copy or replication of data, data actually stored
 - Must be refreshed periodically to match corresponding base tables

A Sample Create View Command

```
CREATE VIEW ExpensiveStuff_V
AS
  SELECT ProductID, ProductDescription, ProductStandardPrice
  FROM Product_T
  WHERE ProductStandardPrice > 300
  WITH CHECK OPTION;
```

- View has a name
- View is based on a SELECT statement
- CHECK_OPTION works only for updateable views and prevents updates that would create rows not included in the view

Advantages of Dynamic Views (1 of 2)

- Simplify query commands
- Assist with data security
- Enhance programming productivity
- Contain most current base table data
- Use little storage space
- Provide customized view for user
- Establish physical data independence

Advantages of Dynamic Views (2 of 2)

- Use processing time each time view is referenced
- May or may not be directly updateable
- As with all SQL constructs, you should use views with discretion

Routines and Triggers

- Routines
 - Program modules that execute on demand
- Functions
 - routines that return values and take input parameters
- Procedures
 - routines that do not return values and can take input or output parameters
- Triggers
 - routines that execute in response to a database event (INSERT, UPDATE, or DELETE)

Figure 6-12 Triggers Contrasted With Stored Procedures (Based on Mullins, 1995)

Procedures and functions are called explicitly. Triggers are event-driven.

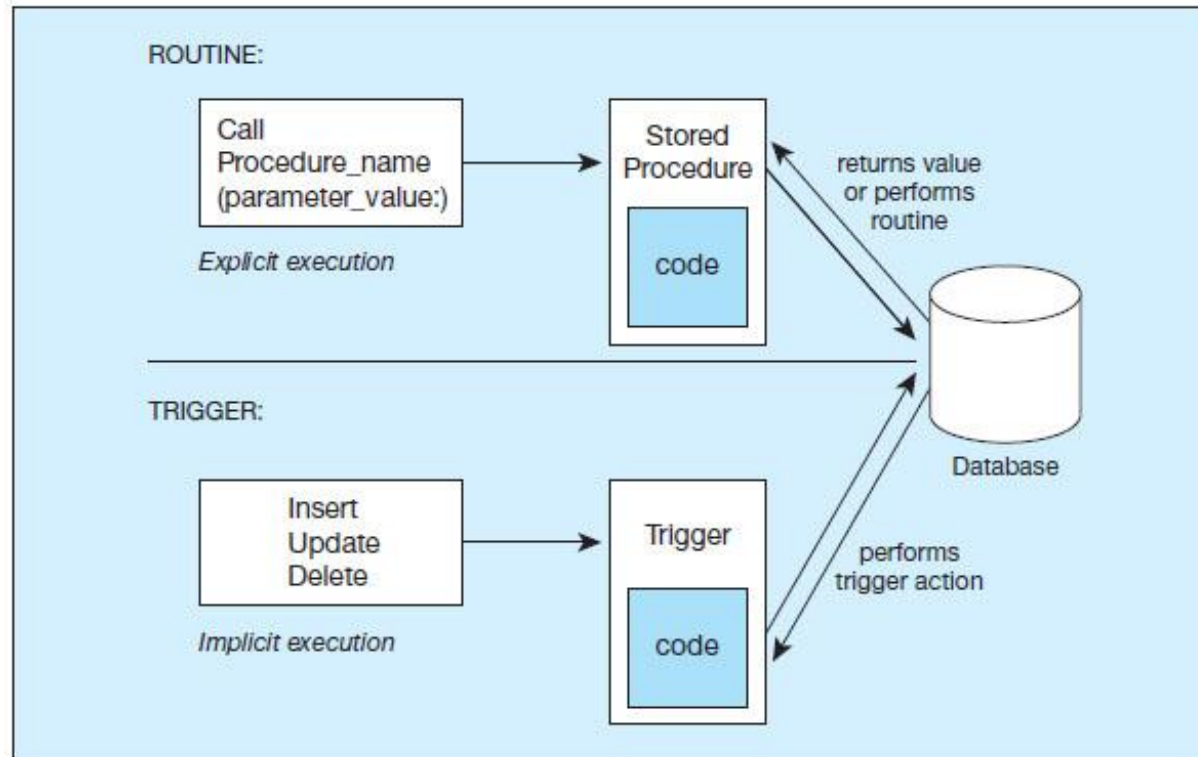


Figure 6-13 Simplified Trigger Syntax in SQL:2008

```
CREATE TRIGGER trigger_name
  {BEFORE| AFTER | INSTEAD OF} {INSERT | DELETE | UPDATE} ON
  table_name
  [FOR EACH {ROW | STATEMENT}] [WHEN (search condition)]
  <triggered SQL statement here>;
```

Example DML trigger

```
CREATE TRIGGER StandardPriceUpdate
AFTER UPDATE OF ProductStandardPrice ON Product_T
FOR EACH ROW
INSERT INTO PriceUpdates_T VALUES (ProductDescription, SYSDATE,
ProductStandardPrice);
```

Example DDL trigger

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
  PRINT 'You must disable Trigger "safety" to drop or alter tables!'
  ROLLBACK;
```

Figure 6-14 Syntax for Creating a Routine in SQL:2011

```
{CREATE PROCEDURE | CREATE FUNCTION} routine_name
([parameter [{parameter} . . .]])
[RETURNS data_type result_cast] /* for functions only */
[LANGUAGE {ADA | C | COBOL | FORTRAN | MUMPS | PASCAL | PLI | SQL}]
[PARAMETER STYLE {SQL | GENERAL}]
[SPECIFIC specific_name]
[DETERMINISTIC | NOT DETERMINISTIC]
[NO SQL | CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA]
[RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT]
[DYNAMIC RESULT SETS unsigned_integer] /* for procedures only */
[STATIC DISPATCH] /* for functions only */
[NEW SAVEPOINT LEVEL | OLD SAVEPOINT LEVEL]
routine_body
```

Example DDL trigger

```
CREATE OR REPLACE PROCEDURE ProductLineSale
AS BEGIN
    UPDATE Product_T
        SET SalePrice = .90 * ProductStandardPrice
        WHERE ProductStandardPrice > = 400;
    UPDATE Product_T
        SET SalePrice = .85 * ProductStandardPrice
        WHERE ProductStandardPrice < 400;
END;
```

Calling the procedure

```
SQL > EXEC ProductLineSale
```

Data Dictionary Facilities

- System tables that store metadata
- Users usually can view some of these tables
- Users are restricted from updating them
- Examples in Oracle 12c
 - DBA_TABLES – descriptions of tables
 - DBA_USERS – information about the users of the system
- Examples in Microsoft SQL Server 2016
 - sys.columns – table and column definitions
 - sys.indexes – table index information

SQL Enhancements/Extensions (1 of 2)

- User-defined data types (UDT)
 - Subclasses of standard types or an object type
- Analytical functions (for OLAP)
 - CEILING, FLOOR, SQRT, RANK, DENSE_RANK, ROLLUP, CUBE, SAMPLE,
 - WINDOW – improved numerical analysis capabilities
- New Data Types
 - BIGINT, MULTISSET (collection), XML
- CREATE TABLE LIKE
 - create a new table similar to an existing one
- MERGE

SQL Enhancements/Extensions (2 of 2)

- Programming extensions
- Persistent Stored Modules (SQL/PSM)
- Capability to create and drop code modules
- New statements: CASE, IF, LOOP, FOR, WHILE, etc.
- Makes SQL into a procedural language
- Oracle has propriety version called PL/SQL, and Microsoft SQL Server has Transact/SQL

Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.